Homomorphic-Policy Attribute-Based Key Encapsulation Mechanisms

Jérémy Chotard, Duong Hieu Phan, David Pointcheval

Homomorphic-Policy Attribute-Based Key Encapsulation Mechanisms

Jérémy Chotard[1,2,3], Duong Hieu Phan[1], and David Pointcheval[2,3]

[1] XLIM, University of Limoges, CNRS
[2] DIENS, École normale supérieure, CNRS, PSL Research University, Paris, France
[3] INRIA

Abstract. Attribute-Based Encryption (ABE) allows to target the recipients of a message according to a policy expressed as a predicate among some attributes. Ciphertext-policy ABE schemes can choose the policy at the encryption time, contrarily to key-policy ABE schemes that specify the policy at the key generation time, for each user.
In this paper, we define a new property for ABE, on top of a ciphertext-policy ABE scheme: homomorphic-policy. A combiner is able to (publicly) combine ciphertexts under different policies into a ciphertext under a combined policy (AND or OR). This allows to specify even much later the policy for a specific ciphertext: the sender encrypts, and the combiner specifies the policy, without knowing the plaintext.
More precisely, using linear secret sharing schemes (LSSS), we design Attribute-Based Key Encapsulation Mechanisms (ABKEM) with our new Homomorphic-Policy property. Technically, by exploiting a specific property in the structure of LSSS matrix, we can show that, given several encapsulations of the same keys under various policies, anyone can derive an encapsulation of the same key under any combination of the policies. As a consequence, from encapsulations under many single attributes, one can build an encapsulation under a complex policy over the attributes.
Similarly to the case of encryption with homomorphic properties, where malleability weakens confidentiality, homomorphic-policy ABE also weakens the security of an ABE when the combiner colludes with legitimate users. On the other hand, homomorphic-policy provides additional flexibility and nice features when one targets some practical application: in Pay-TV, this allows to separate the content providers that can generate the encapsulations of a session key under every attributes, this key being used to encrypt the payload, and the service providers that build the decryption policies according to the subscriptions. The advantage is that the aggregation of the encapsulations by the service providers does not contain any secret information.

1 Introduction

Attribute-Based Encryption (ABE), introduced by Sahai and Waters [16], is a generalization of some advanced primitives such as identity-based encryption [2, 17] and broadcast encryption [6]. It gives a flexible way to define the target

group of people who can receive the message: encryption and decryption can be based on the user's attributes. This primitive was further developed by Goyal *et al.* [9] who introduced two categories of ABE: *ciphertext-policy* attribute-based encryption (CP-ABE) and *key-policy* attribute-based encryption (KP-ABE). In a CP-ABE scheme, the secret key is associated with a set of attributes and the ciphertext is associated with an access policy over the universe of attributes: a user can decrypt a given ciphertext if he holds the attributes that satisfy the access policy underlying the ciphertext. KP-ABE is the dual to CP-ABE in the sense that an access policy is encoded into the users secret key, and a ciphertext is computed with respect to a set of attributes: the ciphertext is decryptable by a user only if the attributes in the ciphertext satisfy the user's access policy.

CP-ABE and KP-ABE consider different scenarios. In KP-ABE, the encryptor has no control over who has access to the data he encrypts. This is the key-issuer who generates and controls the appropriate keys to grant or deny access to the users. In contrast, in CP-ABE, the encryptor is able to decide who should or should not have access to the data that he encrypts. In the applications we target such as Pay-TV, this would mean that the access control is either dynamically managed by the encryptor (with a ciphertext-policy ABE) or statically managed by the key-issuer (with a key-policy ABE), while in real-life a third-party could be in charge of a dynamic policy.

Fine-Grained Access Control. Over the last few years, there has been a lot of progress in constructing secure and efficient ABE schemes from different assumptions and for different settings [1,3,4,7–10,13–16,18], to name a few. The Sahai-Waters' scheme [16] produces ciphertexts decryptable when at least k attributes overlapped between a ciphertext and a private key. While they showed that this primitive is useful for error-tolerant encryption with biometrics, the lack of expressibility limits its applicability when more general policy are required. Fine-grained access control systems [9] facilitate granting differential access rights to a set of users and allow flexibility in specifying the access rights of individual users. Several techniques are known for implementing fine-grained access control. In our work, we focus on fine-grained access control which are expressed by logic formulas and we rely on the standard Linear Secret Sharing Scheme (LSSS) access structures, first considered in the context of ABE by Goyal *et al.* [9].

1.1 Homomorphic-Policy Attribute-Based Key Encapsulation Mechanisms

In KP-ABE, the access policy is controlled at the key generation phase, while in CP-ABE, the access policy is controlled at the message encryption phase. We go a step further in this consideration by postponing the management of the access policy to a later phase and show how one can manage the access policies without knowing any secret nor the content of message.

Previous works on CP-ABE consider classical encryption: the encryptor, taking as input an access policy and a message, produces a corresponding ciphertext. The encryptor thus manages both the access policy and the encryption of the

Fig. 1. Separation of the roles: content provider (C) – access policy manager (A).

original message. This scenario is unavoidable when limiting the access policy as a single atomic attribute characterizing a user's identity (*e.g.,* identity-based encryption) or a target group of users (*e.g.,* identity-based broadcast encryption) because the encryptor needs to know the message to encrypt with the single attribute. However, in the general case, where the access policy is composed from sub-policies via AND and OR operators, the encryption of a message for the whole access policy can be computed from the ciphertexts of the sub-policies, without the knowledge of the original message.

Aiming to this scenario, where a combiner should manage the access policy without knowing to the original message, we need an additional property in ABE: the homomorphic-policy. This property weakens the security of an ABE when the combiner colludes with legitimate users. However, in our practical application (described below), there is no incentive for the combiner to break the scheme. The combiner is indeed involved in the protocol to improve on the flexibility of the access control, and even if it is corrupted, there is no harm for the system, comparing to the scenario where there is no combiner and everything is managed by a unique authority.

Considering Pay-TV, we can now separate the roles of the content provider and of the manager of the access policies (see Figure 1, the left part): the content provider (C) encapsulates the same session key K under each attribute, encrypts the content under this session key K, and provides the encapsulation together with the encrypted content to the manager of the access policies (A). The latter broadcasts the encrypted content, but according to the access policy, it combines the appropriate encapsulations to produce a unique encapsulation, to be broadcast to the users (the recipients (R)). Each authorized user can decrypt this encapsulation (by owning attributes satisfying the access policy) and get the session key to decrypt the content.

We can also envisage another case where the entities C and A are totally independent. To illustrate this, let us assume the manager (A) is a service of video conferencing (see Figure 1, the right part), and the content provider (C) is a client that asks A to organize a meeting with the participants (P). The authorized participants are identified by several attributes. At the moment of the meeting, C secretly gives A the encapsulations of the session key K, under the various attributes, so that it can publicly distribute it according to the appropriate policy to the participants. Only the authorized participants get access to the session K and can participate to the meeting. The manager A does not learn any secret information, and cannot eavesdrop the meeting.

As explained in the above context, the homomorphic-policy property is compatible for key encapsulation rather than for encryption. Technically, we thus need to define Attribute-Based Key Encapsulation Mechanisms (ABKEM) which

encapsulate a session key for an access policy. Then, the combination of two encapsulations of the same session key under two sub-policies into an encapsulation for the composed access policy is completed via the homomorphic-policy property: if we have encapsulations of a session key K under two policies p_1 and p_2, we will be able to produce an encapsulation of the same session key K for the policies $p_1 \vee p_2$ and $p_1 \wedge p_2$. The achievement of an homomorphic-policy ABKEM is the main contribution of this paper. But of course, this is important to keep all the initial properties of an ABE scheme, and namely the collusion-resistance of the final encapsulation.

1.2 Contribution

As explained above, our main contribution is the definition and construction of Homomorphic-Policy Attribute-Based Key Encapsulation Mechanisms (HP-ABKEM). To this aim

- we focus on homomorphic policy and define attribute-based key encapsulation mechanisms (ABKEM).
- we propose homomorphic-policy methods to combine ciphertexts for AND and OR operations on policies.
- Our construction of ABKEM relies on the Lewko-Waters ABE scheme [11], which security holds in the random-oracle model. ABKEM is very convenient to be used with a Data Encapsulation Method (DEM) for practical applications which encrypt large contents or streams of data, such as the case of Pay-TV. We exploit special properties of LSSS for AND and OR operations and transforms them in an efficient way of combining the corresponding encapsulations.
- we then propose an efficient randomization method for making any ciphertext (possibly obtained from the above combinations) statistically indistinguishable from a fresh ciphertext targeting the same policy. This is important for the security of the system.

Putting altogether, our final result gives an HP-ABKEM which is as efficient as the Lewko-Waters ABE system. It is interesting that we get the homomorphic-policy property without paying an extra cost. Actually, the final encapsulation after several combinations turns out to be the same as the one the Lewko-Waters sender would have produced, hence the same security level, and namely the collusion-resistance (in the random-oracle model).

1.3 Our Technique

While the homomorphic property for two group laws over the encrypted messages (usually called *fully homomorphic* property) is quite difficult to achieve. Fortunately, achieving *homomorphic policy* seems much easier and more efficient.

Our technique exploits specific structures of the LSSS-matrix and carries them on the combination of encapsulations. The OR operation is relatively easy

to get, because it essentially corresponds to a concatenation of the encapsulations. However, the AND operation does require a particular property on the LSSS-matrix, that we explain below.

Let us first briefly summarize the general method of constructing an LSSS-based ABE, adapted to an ABKEM. For any policy p, expressed as a logic formula, an LSSS-matrix $\mathbf{A} \in \mathbb{K}^{m \times n}$ is generated such that each line $x \in \{1, \ldots, m\}$ corresponds to an attribute, and from a set of attributes that satisfies the policy p, one can do a linear combination on the corresponding lines of the matrix $\mathbf{A}$ to reconstruct the vector $(1, 0, \ldots, 0)$. One then sets $\vec{v} \leftarrow (s, \$, \ldots, \$)^t$ and the share-vector $\vec{\nu} \leftarrow \mathbf{A} \cdot \vec{v}$ for the secret s, where the vector $\vec{v}$ is completed with random components. A linear combination that reconstructs the vector $(1, 0, \ldots, 0)$ leads to the same linear combination on the share-vector $\vec{\nu} = \mathbf{A} \cdot \vec{v}$ that reconstructs the secret s. One can thus encapsulate each element of the vector $\vec{\nu}$ so that a legitimate user can reconstruct the session key $e(g,g)^s$ in a pairing-friendly setting, thanks to the additive property of the exponents.

Now, from an encapsulation for the policy p_1 of the session key $e(g,g)^{s_1}$ and an encapsulation for the policy p_2 of the session key $e(g,g)^{s_2}$, our objective is to produce an encapsulation for the policy $p_1 \wedge p_2$ of the session key $e(g,g)^{s_1+s_2}$. We first observe a property on the LSSS-matrix: with the LSSS-matrix $\mathbf{A}_1 \in \mathbb{K}^{m \times n}$ associated to the policy p_1 and the LSSS-matrix $\mathbf{A}_2 \in \mathbb{K}^{m \times n}$ associated to the policy p_2, the LSSS-matrix of $\mathbf{A}$ associated to a policy $p_1 \wedge p_2$ is of the following form:

$$\mathbf{A}_\wedge = \begin{bmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ 0 & -\mathbf{A}_2^1 & 0 & -\mathbf{A}_2^* \end{bmatrix}$$

where for any $\mathbf{A}$, we denote $\mathbf{A}^1$ the first column and $\mathbf{A}^*$ the matrix $\mathbf{A}$ without the first column (i.e., $\mathbf{A} = \begin{bmatrix} \mathbf{A}^1 & \mathbf{A}^* \end{bmatrix}$).

Looking at the first and the second column of the matrix $\mathbf{A}_\wedge$, the vector $\mathbf{A}_1^1$ is repeated twice in the upper part, and in the bottom part, the corresponding block is $\begin{bmatrix} 0 & -\mathbf{A}_2^1 \end{bmatrix}$. Therefore, if we put $s_1 + s_2$ and $-s_2$ as the two first components of the vector $\vec{v}$, when combining the resulting share-vector according to the known attributes, the upper part will first lead to the secret $s_1 + s_2 - s_2 = s_1$ and the bottom part will lead to the secret $-s_2$. Consequently, in order to produce the encapsulation of $s_1 + s_2$ under $\mathbf{A}_\wedge$, we only need to combine the encapsulation of s_1 in $\mathbf{A}_1$ and the encapsulation of s_2 in $\mathbf{A}_2$. The resulting share-vector is $\mathbf{A} \cdot (s_1 + s_2, -s_2, \$, \ldots, \$)^t$. However, as one could recover individually the secret $s_1 + s_2$ and $-s_2$ with the appropriate attributes in each sub-policies, but not necessarily for the same user, a collusion attack is possible. We thus need a final randomization step to glue everything together.

1.4 Organization of the Paper

In the next section, we provide a few definitions about linear secrete sharing schemes and attribute-based encryption or key encapsulation. In Section 3, we describe our main contribution, with the notion of homomorphic policy. In Section 4, we give a concrete instantiation of homomorphic-policy attribute-based key encapsulation mechanism. It also details the security analysis.

2 Definitions

2.1 Access Structure

For any application with limited access, one needs to define the *access structure*, which precises which combinations of conditions grant access to the data or to the system.

Definition 1 (Access Structure). *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ be a set of parties (human players or attributes). An access structure in $\mathcal{P}$ is a collection $\mathbb{A} \subseteq 2^{\mathcal{P}} \backslash \{\varnothing\}$. The sets in $\mathbb{A}$ are called the authorized sets, while the others are called unauthorized sets.*

When some minimal sets of parties are required to access the system (but any superset is good too), only monotone access structures make sense, since one can always ignore any supplementary party.

Definition 2 (Monotone Access Structure). *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ be a set of parties and $\mathbb{A}$ an access structure in $\mathcal{P}$. $\mathbb{A}$ is said monotone if, for any subsets $B, C \subseteq \mathcal{P}$, if $B \subseteq C$, when $B \in \mathbb{A}$ then $C \in \mathbb{A}$.*

2.2 Linear Secret Sharing Scheme

In order to control access rights according to a monotone access structure, the use of a secret sharing scheme that spreads the secret key among several players is a classical technique. One must use a secret sharing scheme that just allows authorized sets to reconstruct the secret key. This is even better if the secret key is never fully reconstructed, but just in a virtual way to run the restricted process (such as signature or decryption).

Definition 3 (Secret Sharing Scheme). *A secret sharing scheme over a set of parties $\mathcal{P}$, for an access structure $\mathbb{A}$ over $\mathcal{P}$, allows to share a secret s among the players, with shares $\nu_1, \ldots, \nu_m$ such that:*

- *any set of parties in $\mathbb{A}$ can efficiently reconstruct the secret s from their shares;*
- *any set of parties not in $\mathbb{A}$ has no information about the secret s from their shares.*

A linear secret sharing scheme is quite appropriate to share a secret key in order to run the restricted process in a distributed way, since many cryptographic primitives have such linear properties.

Definition 4 (LSSS). *A Linear Secret Sharing Scheme over a field $\mathbb{K}$ and a set of parties $\mathcal{P}$ is defined by a share-generating matrix $\mathbf{A} \in \mathbb{K}^{m \times n}$ and a labeling map $\rho : \{1, \ldots, m\} \to \mathcal{P}$ according to the access policy $\mathbb{A}$: for any $I \subset \{1, \ldots, m\}$, anyone can efficiently find a vector $\vec{c} \in \mathbb{K}^m$ with support I such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$ if and only if $\rho(I) \in \mathbb{A}$.*

In order to share $s \in \mathbb{K}$, one chooses $v_2, \ldots, v_n \xleftarrow{\$} \mathbb{K}$ and sets $\vec{v} \leftarrow (s, v_2, \ldots, v_n)^t$, then the share-vector is $\vec{\nu} \leftarrow \mathbf{A} \cdot \vec{v}$. One would like to be able to reconstruct s from a few coordinates of this share-vector is $\vec{\nu}$. Being able to find such a vector $\vec{c}$ with support I is equivalent to reconstruct s for the players satisfying $\rho(I)$ only: $\sum_{i \in I} c_i \cdot \nu_i = \sum_{i=1}^{m} c_i \cdot \nu_i = \vec{c}^t \cdot \vec{\nu} = \vec{c}^t \cdot \mathbf{A} \cdot \vec{v} = (1, 0, \ldots, 0) \cdot \vec{v} = s$. To give an example, we can refer to the LSSS proposed by Lewko-Waters [11]. It generates the matrix $\mathbf{A}$ and the map ρ from any monotone policy p that is encoded as a boolean tree, with binary AND and OR gates. One does not need to handle NOT gates, since one only considers monotone policies. It is recalled in the full version [5]. We describe it with matrices in Section 4.3, with the proof in the full version [5].

2.3 Attribute-Based Key Encapsulation Mechanism

In this paper, we extend ABE to Attribute-Based Key Encapsulation Mechanism (ABKEM), where the ciphertext encapsulates a session key, later used to encrypt the payload, in a symmetric way.

Definition 5 (ABKEM). *An attribute-based key encapsulation mechanism over an attribute space $\mathfrak{A}$ is defined by four algorithms:*

- $\mathsf{Setup}(\lambda)$*: Takes as input the security parameter, and outputs the master secret key* msk *and the public key* pk*;*
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, \mathsf{a})$*: Takes as input the master secret key* msk*, the identity* id *of a player, and an attribute* $\mathsf{a} \in \mathfrak{A}$*, to output the private decryption key* $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}}$ *for this attribute* a*;*
- $\mathsf{Encaps}(\mathsf{pk}, p)$*: Takes as input the public key* pk *and a policy p, to output a key K and an encapsulation E of this key;*
- $\mathsf{Decaps}(\mathsf{dk}, E)$*: Takes as input a decryption key and an encapsulation E, to output the encapsulated key K or $\perp$.*

A decryption key will indifferently mean a key $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}}$ for a specific user id and a specific attribute a, or a set $\mathsf{dk}_{\mathsf{id}}^{\mathsf{A}}$ of keys specific to a user id, but for many attributes $\mathsf{a} \in \mathsf{A} \subset \mathfrak{A}$. The correctness property is: for any $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$, $\mathsf{dk}_{\mathsf{id}} = \{\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, \mathsf{a})\}_{\mathsf{a} \in \mathsf{A}}$, and $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$, $\mathsf{Decaps}(\mathsf{dk}_{\mathsf{id}}, E) = K$ if A satisfies the policy p. The main security property is the usual indistinguishability (IND), which should prevent collusions of adaptively chosen players, that can also get decryption keys for adaptively chosen attributes:

Definition 6 (IND for ABKEM). *Let us consider an ABKEM over an attribute space $\mathfrak{A}$. No adversary $\mathcal{A}$ should be able to break the following security game against a challenger:*

- *Initialization: the challenger runs the setup algorithm* $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$*, and provides* pk *to the adversary $\mathcal{A}$;*
- *Key Queries: the adversary $\mathcal{A}$ can ask* KeyGen*-queries, for any* id *and any attribute* a *of its choice to get* $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}}$*;*

- *Challenge: the adversary $\mathcal{A}$ provides a policy p to the challenger that runs $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$, and sets $K_b \leftarrow K$ and K_{1-b} as a random key, for a random bit b. It provides (E, K_0, K_1) to the adversary;*
- *Key Queries: the adversary $\mathcal{A}$ can again ask KeyGen-queries of its choice;*
- *Finalize: the adversary $\mathcal{A}$ outputs its guess b' on the bit b.*

We also define the event Cheat, which means that a user (with some identity id) owns a set of attributes A (the set of all the attributes a asked to a Key Query for id) that satisfies p: in such a case, the adversary can trivially guess b. Hence, we only allow adversaries such that $\Pr[\mathsf{Cheat}] = 0$. We then define $\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) = |2 \times \Pr[b' = b] - 1|$, and say that an ABKEM is (t, ε)-adaptively secure if no adversary $\mathcal{A}$ running within time t can get $\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) \geq \varepsilon$.

We stress that everything is adaptive in this definition: the identities and the attributes asked to the key queries, and the policy asked for the challenge query. However, we are in the chosen-plaintext scenario, without access to any decryption/decapsulation oracle.

3 Homomorphic-Policy

3.1 Definition

While CP-ABE allows to specify the policy at the encryption time, which is also the case for our definition of ABKEM, the sender may not be aware of the policy yet. We thus suggest to exploit an homomorphic property on the policy: we would like to allow the derivation of an encapsulation of K under a combination $p = p_1 \wedge p_2$ or $p = p_1 \vee p_2$ from the encapsulations of K under the policies p_1 and p_2 on the attributes in $\mathfrak{A}$, without knowing K (which has already been used to encrypt the payload).

With such an homomorphism on the policies, from the encapsulations of a common key K under all the attributes $\mathsf{a} \in \mathfrak{A}$, one could publicly generate an encapsulation of K under any policy on $\mathfrak{A}$: as illustrated on Figure 2, from the encapsulations $\{E_i\}_i$ of K for the attributes $\mathfrak{A} = \{\mathsf{a}_i\}$, one can derive the encapsulation E_p of K under any policy p, encoded as a binary tree with AND ($\wedge$) and OR ($\vee$) gates. Again, we only consider monotone policies, hence the absence of NOT gates. On attributes, if one wants to consider the negation (or absence) of some attribute a, one has to define a second attribute a' that is exclusive with a, so that, if $p = (\mathsf{a})$, then $\neg p = (\mathsf{a}')$.

To achieve this goal, we just need to be able to combine two encapsulations of K under p_1 and p_2 in order to derive the encapsulation of K under $p_\vee = p_1 \vee p_2$ and under $p_\wedge = p_1 \wedge p_2$. The global encapsulation under a more general policy can then be recursively built.

Definition 7 (HP-ABKEM). *An homomorphic-policy attribute-based key-encapsulation mechanism over an attribute space $\mathfrak{A}$ is an ABKEM (see Definition 5), with a more specific encapsulation algorithm and two additional algorithms for the homomorphism:*

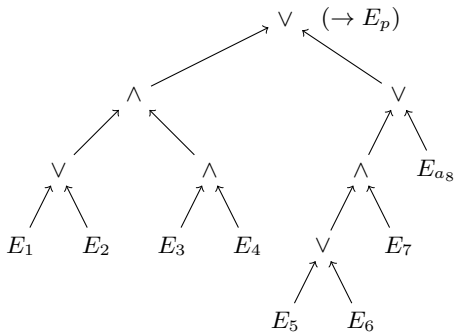

Fig. 2. Derivation of E_p from $\{E_i\}$, for $p = ((\mathsf{a}_1 \vee \mathsf{a}_2) \wedge (\mathsf{a}_3 \wedge \mathsf{a}_4)) \vee (((\mathsf{a}_5 \vee \mathsf{a}_6) \wedge \mathsf{a}_7) \vee \mathsf{a}_8)$

- $\mathsf{Encaps}(\mathsf{pk}, P)$*: Takes as input the public key* pk*, a list of policies* $P = (p_i)_i$*, to output a key* K *and the encapsulations* E_i *of this key under the policies* p_i*'s;*
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{gate}, E_1, E_2)$*: Takes as input the public key* pk *as well as two encapsulations* E_1 *and* E_2*, and a gate* $\mathsf{gate} \in \{\wedge, \vee\}$*, to output an encapsulation under the combination of the initial policies for* E_1 *and* E_2*;*
- $\mathsf{Rand}(\mathsf{pk}, E)$ *Takes as input the public key* pk *as well as an encapsulation, to output a new encapsulation (of the same key under the same policy).*

The intuition behind the new Encaps algorithm is that we want to be able to encapsulate the same key K under various policies. We thus opt for an encapsulation algorithm that takes as input all the policies that will be combined later. The correctness properties are:

- if $(E_i)_i \leftarrow \mathsf{Encaps}(\mathsf{pk}, (p_i)_i)$ are common encapsulations of a key K under the p_i's, then for any i, j, $E \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{gate}, E_i, E_j)$ is an encapsulation of the same key K, but under the policy $p = p_i \ \mathsf{gate} \ p_j$;
- for any encapsulation E of some key K under a policy p, $E' \leftarrow \mathsf{Rand}(\mathsf{pk}, E)$ follows the same distribution as a fresh encapsulation of K under the policy p.

Note that we do not expect the combination to hide the structure of the initial encapsulations. The randomization will do this work, but there is no need to do it at each step, hence the separation of the two processes: one will iteratively combine the encapsulations in order to obtain the encapsulation under the appropriate policy, and then the randomization will finalize the process. Figure 3 illustrates this fact: combining and randomizing at each step leads to exactly the same distribution of the root encapsulation as combining at each step and randomizing at the last step only.

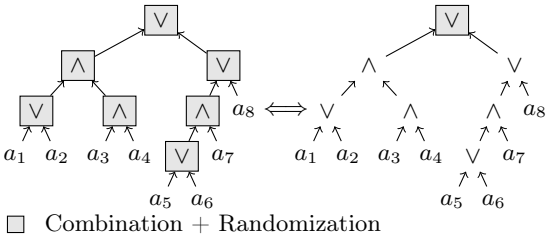

Fig. 3. Randomization Process in Combination

3.2 Security

As explained in the Pay-TV scenario in the introduction, we have three players: the content provider (or the sender), the manager of the access policy (or the combiner) and the receiver. We thus expect the sender to encapsulate a key K under each attribute, and to encrypt the payload under K; the combiner then generates the encapsulation of K under the appropriate policy; so that only the legitimate receivers can decapsulate and decrypt the payload.

When the adversary plays the role of the receivers, the required security notion is exactly the previous indistinguishability: given several keys for various attributes, and even several identities (to model collusions), an adversary should not be able to get any information about a key encapsulated under a policy that is not satisfies by any of the users under its control. We stress that this indistinguishability game (IND) models the resistance against the collusion of receivers. But both the sender and the combiner are considered honest.

On the other hand, the sender may not totally trust the combiner and may want to limit the risk in case the combiner would be corrupted: while the former sends K encapsulated under many attributes (or more generally many policies), the latter should not be able to distinguish K from a random key, in order to guarantee to privacy of the content encrypted under K. Hence the new indistinguishability game with multiple encapsulations ($\mathsf{m}-\mathsf{IND}$), but without being able to get any decryption key, hence the no-key attack (NKA). Since the adversary does not have access to any decryption key, this security scenario does not allow the combiner to collude with anybody, and namely not with any receiver.

Definition 8 ($\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ **for ABKEM).** *Let us consider an ABKEM over an attribute space $\mathfrak{A}$. No adversary $\mathcal{A}$ should be able to break the following security game against a challenger:*

- *Initialization: the challenger runs the setup algorithm* $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$*, and provides* pk *to the adversary $\mathcal{A}$;*
- *Challenge: the challenger runs* $(K, (E_i)_i) \leftarrow \mathsf{Encaps}(\mathsf{pk}, \mathfrak{A})$*, and sets* $K_b \leftarrow K$ *and* K_{1-b} *as a random key, for a random bit b. It provides* $((E_i)_i, K_0, K_1)$ *to the adversary;*
- *Finalize: the adversary $\mathcal{A}$ outputs its guess b' on the bit b.*

We then define $\mathsf{Adv}^{\mathsf{m-ind-nka}}(\mathcal{A}) = |2 \times \Pr[b' = b] - 1|$*, and say that an ABKEM is* (t, ε)-$\mathsf{m-IND}$ *if no adversary* $\mathcal{A}$ *running within time* t *can get* $\mathsf{Adv}^{\mathsf{m-ind-nka}}(\mathcal{A}) \geq \varepsilon$.

We stress that now, nothing is adaptive, since the adversary cannot get decryption keys, but gets the encapsulations of the same key K under all the individual attributes. We also remain in the chosen-plaintext scenario, without access to any decryption/decapsulation oracle. In addition, since the adversary is the combiner that receives the key K encapsulated under every attribute, we do not allow any collusion with a user: any attribute would be enough to get K and break the security game.

On can note that in the real-life, such a combiner would not be a critical party since it does not know any long-term secret. Of course, it will learn ephemeral encapsulations that would allow any receiver (with attributes that satisfy the final policy or not) to decapsulate the session key and to decrypt the content. But a short-term corruption will just leak the content during a short period, and not for ever.

4 Construction

4.1 Modified Lewko-Waters Scheme

We present here a revised version of the ABE scheme from [11]. First, for the sake of simplicity, we do not exploit the decentralized version and so all the attributes are managed by the same entity (but we could keep the decentralized version). Second, for the homomorphic property, we consider a Key Encapsulation Mechanism (KEM) instead of an encryption scheme, which just encaps a session key. However, we still use an LSSS to realize the access policy and pairing techniques to ensure collusion resistance. More precisely, we use a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$, where the groups $\mathbb{G}$ and $\mathbb{G}_T$ will be of composite order $N = q_1 q_2 q_3$, with three large prime integers q_1, q_2, and q_3. Let us first describe our variant of ABKEM.

4.2 Description

- $\mathsf{Setup}(\lambda)$: One first generates a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ for groups of composite order $N = q_1 q_2 q_3$ (of length λ). One also generates a generator g_1 of the subgroup $\mathbb{G}_1 \subset \mathbb{G}$ of order q_1 and a hash function $\mathcal{H} : \{0,1\}^* \longrightarrow \mathbb{G}$. We also denote $G = e(g_1, g_1) \in \mathbb{G}_T$. Then, for each attribute a, the authority specifies the pair of secret/public keys, respectively $\mathsf{sk_a} = (\alpha_\mathsf{a}, y_\mathsf{a})$ and $\mathsf{pk_a} = (G_\mathsf{a} = G^{\alpha_\mathsf{a}}, g_\mathsf{a} = g_1^{y_\mathsf{a}})$. The master secret key msk is the concatenation of the $\mathsf{sk_a}$'s, and the public key pk contains N, g_1 and $\mathcal{H}$, together with the concatenation of the $\mathsf{pk_a}$'s.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, \mathsf{a})$: From $\mathsf{msk} = \{\mathsf{sk_a}\}$, id and a, the authority outputs $\mathsf{dk}^\mathsf{a}_\mathsf{id} = g_1^{\alpha_\mathsf{a}} \mathcal{H}(\mathsf{id})^{y_\mathsf{a}}$.

- $\mathsf{Encaps}(\mathsf{pk}, P)$: From the public key pk and a set P of policies, one first chooses some random $s \xleftarrow{\$} \mathbb{Z}_N$ and sets the symmetric encapsulated key $K \leftarrow G^s$. Then, for each $p \in P$, we process the following encapsulation: from the LSSS matrix $\mathbf{A} \in \mathbb{K}^{m \times n}$ and the associated labeling map ρ onto the attributes describing the access structure defined by the policy p, we set $\vec{v} = (s, v_2, \ldots, v_n)$ and $\vec{w} = (0, w_2, \ldots, w_n)$, with $v_k, w_k \xleftarrow{\$} \mathbb{Z}_N$ for $k = 2, \ldots, n$ and $\vec{r} \xleftarrow{\$} \mathbb{Z}_N^m$. We build the share vectors $\vec{\nu} = \mathbf{A} \cdot \vec{v}$ and $\vec{\omega} = \mathbf{A} \cdot \vec{w}$. Eventually, for each line $x \in \{1, \ldots, m\}$ of the matrix $\mathbf{A}$, we construct the encapsulation using the keys $\mathsf{pk}_{\rho(x)} = (G_{\rho(x)}, g_{\rho(x)})$ associated to the attribute $\mathsf{a}_x = \rho(x)$ involved in the policy p:

$$E_{1,x} = G^{\nu_x} \cdot G_{\rho(x)}^{r_x} \qquad E_{2,x} = g_1^{r_x} \qquad E_{3,x} = g_1^{\omega_x} \cdot g_{\rho(x)}^{r_x}$$

  The algorithm returns $E_p = \{(E_{1,x}, E_{2,x}, E_{3,x})\}_x$ for each $p \in P$.
- $\mathsf{Decaps}(\mathsf{dk}_{\mathsf{id}}, E_p)$, where $\mathsf{dk}_{\mathsf{id}} = (\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}})$ for the attributes owned by id: First, the user must find a vector $\vec{c} \in \mathbb{K}^m$ such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$ and the support I of the non-zero components of $\vec{c}$ links to a set of attributes owned by the user. Then, for each $x \in I$, the user computes $F_x = E_{1,x} \cdot e(H(\mathsf{id}), E_{3,x}) / e(\mathsf{dk}_{\mathsf{id}}^{\rho(x)}, E_{2,x})$. He finally gets K by combining with the vector $\vec{c}$: $K \leftarrow \prod_{x \in I} F_x^{c_x}$.

The latter reconstruction works since

$$\sum_{x \in I} c_x \cdot \nu_x = \sum_{x=1}^{m} c_x \cdot \nu_x = \vec{c}^t \cdot \vec{\nu} = \vec{c}^t \cdot \mathbf{A} \cdot \vec{v} = (1, 0, \ldots, 0) \cdot \vec{v} = s$$

$$\sum_{x \in I} c_x \cdot \omega_x = \sum_{x=1}^{m} c_x \cdot \omega_x = \vec{c}^t \cdot \vec{\omega} = \vec{c}^t \cdot \mathbf{A} \cdot \vec{w} = (1, 0, \ldots, 0) \cdot \vec{w} = 0$$

In addition, for each $x \in I$,

$$F_x = E_{1,x} \cdot e(H(\mathsf{id}), E_{3,x}) / e(\mathsf{dk}_{\mathsf{id}}^{\rho(x)}, E_{2,x}) = G^{\nu_x} e(H(\mathsf{id}), g_1)^{\omega_x}.$$

And so, the final combination leads to

$$\prod_{x \in I} F_x^{c_x} = \prod_{x \in I} (G^{\nu_x} e(H(\mathsf{id}), g_1)^{\omega_x})^{c_x} = G^{\vec{c}^t \cdot \vec{\nu}} \cdot e(H(\mathsf{id}), g_1)^{\vec{c}^t \cdot \vec{\omega}} = G^s.$$

One should note that for this construction to work, the map ρ needs to be an injection. In practice, this is not a real issue, since one can simply duplicate the attributes and provide multiple keys to users.

4.3 Construction of the LSSS

In this section, we detail a construction of the LSSS, in an iterative way, from a boolean tree (with only OR and AND gates).

First, we have to start from an LSSS for a simple policy $p = (\mathsf{a}_i)$, for some i (i.e., a unique attribute): $\mathbf{A}_i = (1)$ and $\rho(1) = i$. Then we explain how to combine two policies p_1 and p_2, represented by the LSSS's $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$ respectively, into the policies $p_\wedge = p_1 \wedge p_2$ and $p_\vee = p_1 \vee p_2$ with LSSS's $(\mathbf{A}_\wedge, \rho_\wedge)$ and $(\mathbf{A}_\vee, \rho_\vee)$ respectively.

In the following, for any $\mathbf{A}$, we denote $\mathbf{A}^1$ the first column et $\mathbf{A}^*$ the matrix $\mathbf{A}$ without the first column (i.e., $\mathbf{A} = \left[\mathbf{A}^1 \ \mathbf{A}^*\right]$).

Proposition 9. *Let $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$ be two LSSS's for the policies p_1 and p_2. Then we can build the LSSS's $(\mathbf{A}_\wedge, \rho_\wedge)$ and $(\mathbf{A}_\vee, \rho_\vee)$ for the policies $p_\wedge = p_1 \wedge p_2$ and $p_\vee = p_1 \vee p_2$ as follows*

$$\mathbf{A}_\vee = \begin{bmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ \mathbf{A}_2^1 & 0 & \mathbf{A}_2^* \end{bmatrix} \quad \mathbf{A}_\wedge = \begin{bmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ 0 & -\mathbf{A}_2^1 & 0 & -\mathbf{A}_2^* \end{bmatrix}$$

If we label the rows of the matrices from 1 to $m_1 + m_2$, where $\mathbf{A}_1 \in \mathbb{K}^{m_1 \times n_1}$ and $\mathbf{A}_2 \in \mathbb{K}^{m_2 \times n_2}$, we have

$$\rho_\wedge = \rho_\vee : x \mapsto \begin{cases} \rho_1(x), \text{ if } x \leq m_1 \\ \rho_2(x - m_1), \text{ if } x \geq m_1 + 1 \end{cases}$$

This construction is not really new, since it was described in [12] in a more generic way. But we need this explicit description for the security analysis of our ABKEM. The correctness of this LSSS construction is provided in the full version [5]. Up to a re-ordering of the rows and columns of the matrices, this is also the same construction obtained from the algorithm presented in the full version [5] from [11]. A comparison of the two methods is indeed proposed in the full version [5].

4.4 Homomorphic Policy

Our main goal is now to show that this iterative construction of the LSSS can be applied to our ABKEM, starting from encapsulations of the same key K under every attribute. This will follow from the homomorphic-policy property.

We recall that in the ABKEM, $\vec{\nu} = \mathbf{A} \cdot \vec{v}$ is a secret sharing of a random scalar s, while $\vec{\omega} = \mathbf{A} \cdot \vec{w}$ is a secret sharing of 0, the components ν_x and ω_x being hidden in $E_{1,x}$ and $E_{3,x}$ by $G_{\rho(x)}^{r_x}$ and $g_{\rho(x)}^{r_x}$ respectively. Because of the linear property of the LSSS, by concatenating or by adding the shares, we either obtain the OR or the AND policies of two encapsulations $E^{(1)}$ and $E^{(2)}$:

Share-Vectors		**Encapsulations**
$\begin{bmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{bmatrix}$	$\longleftrightarrow$	$E^{(1)} \cup E^{(2)}$
$\vec{\nu}_1 + \vec{\nu}_2$	$\longleftrightarrow$	$E^{(1)} \cdot E^{(2)}$

Of course, the same applies on the shares $\vec{\omega}$ of 0, but we focus on the shares $\vec{\nu}$ of the random s

One Secret under two Policies. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of the same secret value $K = G^s$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$.

The construction thus used the share-vectors $\vec{\nu}_i = (\nu_{i,1}, \ldots, \nu_{i,m_i}) = \mathbf{A}_i \cdot \vec{v}_i$, with $\vec{v}_i = (s, v_{i,2}, \ldots, v_{i,n_i})^t$, for $i = 1, 2$. Using

$$\mathbf{A}_\vee = \begin{bmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ \mathbf{A}_2^1 & 0 & \mathbf{A}_2^* \end{bmatrix} \text{ and } \vec{v} = (s, v_{1,2}, \ldots, v_{1,n_1}, v_{2,2}, \ldots, v_{2,n_2})^t,$$

one gets $\vec{\nu} = \begin{bmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{bmatrix}$.

From attributes satisfying p_i, under the LSSS property, one can efficiently find a vector $\vec{c}_i = (c_{i,1}, \ldots, c_{i,m_i})^t \in \mathbb{K}^m$ such that $\vec{c}_i^t \cdot \mathbf{A}_i = (1, 0, \ldots, 0)$. By multiplying this vector on the appropriate half of $\vec{\nu}$, one can get s:

$$(c_{1,1}, \ldots, c_{1,m_1}, 0, \ldots, 0) \cdot \vec{\nu} = \vec{c}_1^t \cdot \vec{\nu}_1 = s$$
$$(0, \ldots, 0, c_{2,1}, \ldots, c_{2,m_2}) \cdot \vec{\nu} = \vec{c}_2^t \cdot \vec{\nu}_2 = s.$$

It will be used for the disjunction of policies.

Two Secrets under different Policies. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of two secret values $K_1 = G^{s_1}$ and $K_2 = G^{s_2}$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$.

The construction thus used the share-vectors $\vec{\nu}_i = (\nu_{i,1}, \ldots, \nu_{i,m_i}) = \mathbf{A}_i \cdot \vec{v}_i$, with $\vec{v}_i = (s_i, v_{i,2}, \ldots, v_{i,n_i})^t$, for $i = 1, 2$. Using

$$\mathbf{A}_\wedge = \begin{bmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ 0 & -\mathbf{A}_2^1 & 0 & -\mathbf{A}_2^* \end{bmatrix} \text{ and } \vec{v} = (s_1 + s_2, -s_2, v_{1,2}, \ldots, v_{1,n_1}, v_{2,2}, \ldots, v_{2,n_2})^t,$$

one gets again $\vec{\nu} = \begin{bmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{bmatrix}$. This combination will be used for the conjunction of policies, but only with the same secret. Note that the produced encapsulation must be randomized to perform the new policy, otherwise there is a colluding attack: with independent keys for each policy, two players can independently get s_1 and s_2, and can then combine them to get $s_1 + s_2$.

Two Secrets under the same Policy. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of two secret values $K_1 = G^{s_1}$ and $K_2 = G^{s_2}$ under the same policy p, represented by the LSSS $(\mathbf{A}, \rho)$.

The construction thus used the share-vectors $\vec{\nu}_1$ and $\vec{\nu}_2$ of the random scalars s_1 and s_2 respectively under the same policy p. Then, one can see $\vec{\nu} = \vec{\nu}_1 + \vec{\nu}_2$ as a share-vector of $s = s_1 + s_2$ under the policy p, since $\vec{\nu} = \mathbf{A} \cdot (\vec{v}_1 + \vec{v}_2)$. Indeed, from attributes satisfying p, one can efficiently find a vector $\vec{c} \in \mathbb{K}^m$ such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$:

$$\vec{c}^t \cdot \vec{\nu} = \vec{c}^t \cdot \mathbf{A} \cdot (\vec{v}_1 + \vec{v}_2) = (1, 0 \ldots, 0) \cdot (\vec{v}_1 + \vec{v}_2) = s_1 + s_2.$$

This combination will be used for the randomization, with $s_2 = 0$.

4.5 Security

IND *security.* In [11], Lewko and Waters proved their ABE scheme to be indistinguishable under several assumptions in the composite-order pairing setting (we recall them in the full version [5]) and the condition that ρ is injective. This easily leads to the IND security for the above variant of ABKEM, even for adaptive KeyGen-queries. Hence, this ABKEM construction achieves the IND security level.

$\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ *security.* We now show that, the $\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ security of the modified ABKEM can also be based on the IND security of Lewko-Waters scheme.

Theorem 10. *The* IND *security level of Lewko-Waters implies the* $\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ *security of the modified ABKEM.*

Proof. As highlighted in the full version [5], the two security games are quite similar, the main differences appear in the challenge phase, and the lack of key-queries in the latter. If one looks at the above construction of the LSSS-matrix, for $p = \mathsf{a}_1 \vee \ldots \vee \mathsf{a}_k$, then $\mathbf{A} = (1, \ldots, 1)^t$ and $\vec{\nu} = (s, \ldots, s)^t$: from an encapsulation E of the key $K = G^s$ under the policy p, one can easily extract the encapsulations E_i of the same K, under the policies $p_i = (\mathsf{a}_i)$ respectively: indeed, each triple $(E_{1,x}, E_{2,x}, E_{3,x})$ is a simple encapsulation of K under $\mathsf{a}_x = \rho(x)$.

This remark is true for every conjunction $p_f = \bigvee p_i$ where the policies p_i's do not share any attribute. Note that the triples $(E_{1,x}, E_{2,x}, E_{3,x})$ involved in the decryption of a policy p_i are those associated to the attributes which appears in this policy. The choice of these triples is given by the vector c. Consequently, we can easily convert the challenger's answer from one game to another by concatenating/separating the ciphertext(s) by following this policy decomposition. Because of the lack of key-queries in the $\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ security game, we can just build an adversary $\mathcal{B}$ for the IND game from an adversary $\mathcal{A}$ of the $\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ game. More precisely, if an adversary $\mathcal{A}$ has an advantage $\mathsf{Adv}^{\mathsf{m-ind-nka}}(\mathcal{A}) = \varepsilon$ in the $\mathsf{m}-\mathsf{IND}-\mathsf{NKA}$ game for the policies $(p_j)_j$, one can construct an adversary $\mathcal{B}$ with the same advantage $\mathsf{Adv}^{\mathsf{ind}}(\mathcal{B}) = \varepsilon$ in the IND game for the policy $p_f = \bigvee p_i$.

As already noted, Lewko and Waters [11] assume a one-use restriction on attributes throughout the proof: this means that the row-labeling map ρ of the challenge ciphertext access matrix $(\mathbf{A}, \rho)$ must be injective. The reason is that, if an attribute is used twice in the access matrix, then there will appear an implicit relation between the randomnesses associated to the corresponding two lines of the matrix and the proof does not go through anymore. To overcome this issue, Lewko and Waters suggested to associate k independent attributes to any attribute a, where k is an upper-bound on the number of repetitions of an attribute in a policy. Our scheme inherently has the same limitation.

4.6 Homomorphic Policy

Let us now see how this impacts on the encapsulations, when one wants to do disjunctions and conjunctions of policies.

Disjunctions. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of the same key $K = G^s$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$. We want to make an encapsulation of K under the policy $p_1 \vee p_2$. Using the construction of the share-vectors from Section 4.4, which applies on both $\vec{\nu}_1, \vec{\nu}_2$ and $\vec{\omega}_1, \vec{\omega}_2$, we know that the resulting encapsulation should use

$$\vec{\nu} = \begin{bmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{bmatrix} \quad \vec{\omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vec{\omega}_2 \end{bmatrix}.$$

Therefore, the resulting encapsulation is $E_{p_1 \vee p_2} = \{(E^{(1)}_{j,x}, E^{(2)}_{j,x})_{j=1,2,3}\}_{x \in \mathfrak{A}}$.

Conjunctions. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of the same key $K = G^s$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$. We want to make an encapsulation of K under the policy $p_1 \wedge p_2$. Using the construction of the share-vectors from Section 4.4, which applies on both $\vec{\nu}_1, \vec{\nu}_2$ and $\vec{\omega}_1, \vec{\omega}_2$, we know that the resulting encapsulation should use

$$\vec{\nu} = \begin{bmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{bmatrix} \quad \vec{\omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vec{\omega}_2 \end{bmatrix}.$$

However, this will contain the key $K^2 = G^{2s}$. We thus have to use square-roots: the resulting encapsulation is $E_{p_1 \wedge p_2} = \{((E^{(1)}_{j,x})^{1/2}, (E^{(2)}_{j,x})^{1/2})_{j=1,2,3}\}_{x \in \mathfrak{A}}$.

Note that even if in the Lewko-Waters' construction there is a modulus $N = q_1 q_2 q_3$ that is hard to factor, this is the order of the group. Hence $g^{1/2} = g^{\alpha}$ where $\alpha = (N+1)/2$.

As already noted, collusion is possible. But this is even worse in this case since we are using $s = s_1 = s_2$: just satisfying one of the two policies, one can recover $K^{1/2} = G^{s/2}$, which thereafter easily leads to K. We thus need to randomize the encapsulation, in order to glue together the policies.

Randomization. If one looks in details the description of the Encaps algorithm, there are 4 kinds of randomness:

- s, that defined the encapsulated key $K = G^s$;
- $v_k, w_k \xleftarrow{\$} \mathbb{Z}_N$ for $k = 2, \ldots, n$, to define $\vec{v}$ and $\vec{w}$;
- $\vec{r} \xleftarrow{\$} \mathbb{Z}_N^m$.

Let us start from any encapsulation $E^{(1)}$ of K under a policy p, with

$$E^{(1)}_{1,x} = G^{\nu^{(1)}_x} \cdot G^{r^{(1)}_x}_{\rho(x)} \qquad E^{(1)}_{2,x} = g_1^{r^{(1)}_x} \qquad E^{(1)}_{3,x} = g_1^{\omega^{(1)}_x} \cdot g^{r^{(1)}_x}_{\rho(x)}$$

for each $\mathsf{a}_x = \rho(x)$ involved in the policy p, where $\vec{\nu}^{(1)} = \mathbf{A} \cdot \vec{v}^{(1)}$ and $\vec{\omega}^{(1)} = \mathbf{A} \cdot \vec{w}^{(1)}$. We now define a new fresh encapsulation $E^{(2)}$:

$$E^{(2)}_{1,x} = G^{\nu^{(2)}_x} \cdot G^{r^{(2)}_x}_{\rho(x)} \qquad E^{(2)}_{2,x} = g_1^{r^{(2)}_x} \qquad E^{(2)}_{3,x} = g_1^{\omega^{(2)}_x} \cdot g^{r^{(2)}_x}_{\rho(x)}$$

where $\vec{\nu}^{(2)} = \mathbf{A} \cdot \vec{v}^{(2)}$ and $\vec{\omega}^{(2)} = \mathbf{A} \cdot \vec{w}^{(2)}$, for $\vec{v}^{(2)} = (0, v'_2, \ldots, v'_n)^t$ and $\vec{w}^{(2)} = (0, w'_2, \ldots, w'_n)^t$, with $v'_k, w'_k \xleftarrow{\$} \mathbb{Z}_N$ for $k = 2, \ldots, n$, and $\vec{r}^{(2)} \xleftarrow{\$} \mathbb{Z}_N^m$. This is actually a fresh random encapsulation of $K^{(2)} = 1_{\mathbb{G}_T}$ under the policy p. It can be computed from the public key pk that contains N, g_1, and the keys $\mathsf{pk}_\mathsf{a} = (G_\mathsf{a}, g_\mathsf{a})$, for all the attributes, as would be generated a fresh encapsulation of $K = 1_{\mathbb{G}_T}$.

Eventually, the new encapsulation $E = \{(E^{(1)}_{1,x} \cdot E^{(2)}_{1,x}, E^{(1)}_{2,x} \cdot E^{(2)}_{2,x}, E^{(1)}_{3,x} \cdot E^{(2)}_{3,x})\}_x$ is a truly random encapsulation of the same K under the policy p, and so looks like a fresh encapsulation.

Conclusion

We proposed a new feature for ABE, with the homomorphic policy. It allows to separate the roles of the sender and the access right manager. This is a quite useful property for the Pay-TV context, since the access right manager does not have access anymore to the content payload. The distribution to the subscribers can be performed by a weakly trusted party.

Acknowledgments.

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud) and by the French ANR ALAMBIC project (ANR-16-CE39-0006).

References

1. N. Attrapadung, B. Libert, and E. de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 90–108. Springer, Heidelberg, Mar. 2011.
2. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, Aug. 2001.
3. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

4. C. Chen, J. Chen, H. W. Lim, Z. Zhang, D. Feng, S. Ling, and H. Wang. Fully secure attribute-based systems with short ciphertexts/signatures and threshold access structures. In E. Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 50–67. Springer, Heidelberg, Feb. / Mar. 2013.
5. J. Chotard, D. H. Phan, and D. Pointcheval. Homomorphic-policy attribute-based key encapsulation mechanisms. Cryptology ePrint Archive, Report 2016/1089, 2016. `http://eprint.iacr.org/2016/1089`.
6. A. Fiat and M. Naor. Broadcast encryption. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, Aug. 1994.
7. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.
8. V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded ciphertext policy attribute based encryption. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 579–591. Springer, Heidelberg, July 2008.
9. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 89–98. ACM Press, Oct. / Nov. 2006. Available as Cryptology ePrint Archive Report 2006/309.
10. J. Herranz, F. Laguillaumie, and C. Ràfols. Constant size ciphertexts in threshold attribute-based encryption. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 19–34. Springer, Heidelberg, May 2010.
11. A. B. Lewko and B. Waters. Decentralizing attribute-based encryption. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588. Springer, Heidelberg, May 2011.
12. V. Nikov and S. Nikova. New monotone span programs from old. Cryptology ePrint Archive, Report 2004/282, 2004. `http://eprint.iacr.org/2004/282`.
13. T. Okamoto and K. Takashima. Fully secure unbounded inner-product and attribute-based encryption. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 349–366. Springer, Heidelberg, Dec. 2012.
14. R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 195–203. ACM Press, Oct. 2007.
15. Y. Rouselakis and B. Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 463–474. ACM Press, Nov. 2013.
16. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.
17. A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, Aug. 1984.
18. S. Yamada, N. Attrapadung, G. Hanaoka, and N. Kunihiro. A framework and compact constructions for non-monotonic attribute-based encryption. In H. Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 275–292. Springer, Heidelberg, Mar. 2014.